Comparison of use of Machine Learning Methods for Signal Classification Problem

Wylie Standage-Beier
Arizona State University
Tempe, AZ
wstandag@asu.edu

Muslum Emir Avci
Arizona State University
Tempe, AZ
mavci@asu.edu

Muhammad Bilal
Arizona State University
Tempe, AZ
mbilal2@asu.edu

***Abstract*—In this project, we explore the different machine learning (ML) and deep learning (DL) methods that are used for classifying a signal's modulation scheme. We explore two DL methods that are used in current literature and compare their results. First DL method we used is just a simple convolutional neural network (CNN) that extracts one feature set from two different signal channels (in-phase and quadrature). In the second DL method we investigate develop a technique to also includes the relation between I and Q channels. Our results show that, accounting the relation between I and Q signals improves performance by two percent compared to simple CNN. Which points, there is information between I/Q signals that would improve the accuracy of the classification.**

***Index Terms*—signal classification, telecommunications, modulation, cognitive radio, deep learning, ML**

I. Introduction

A. Problem Statement

Knowledge of the modulation being used is usually build into the communications systems. However, in the absence of this assumption the modulation must be inferred. The determination of communications signals' modulation from a set of modulations by observation is the problem being studied. High level overview as shown by Fig. 1 is applying ML to determination of the signal's modulation. The radio signal is captured and stored using a Software Defined Radio (SDR) and processed to determine the captured signals modulation. Machine learning is used to make this inference.

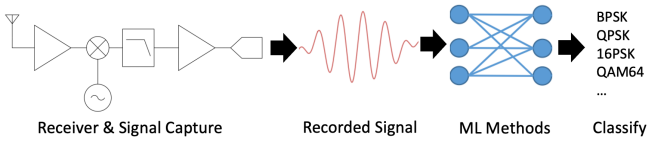


Fig. 1. Project Overview and Block Diagram

B. Background Information

In a wireless communication system, information is encoded into transmitted electromagnetic wave in two steps. First step is forming a baseband signal that contains the actual information, such as a message, picture or any other data type using a predetermined modulation scheme. Frequency content of the baseband signal is generally much smaller than the actual transmitted wave and it is generally not feasible to transmit baseband signal directly. Therefore, in the second step, baseband signal is upconverted into RF frequencies using a mixer, where information could be transmitted Over The Air (OTA) by an antenna. Receiver generally performs this action in reverse by first downconverting signal into baseband and then processing this signal using an analog-to-digital converter (ADC), and digital signal processing (DSP). To preserve the information content of the signal, RF signal is downconverted into two baseband signals as shown in Fig. 2. First baseband signal is generated by multiplying RF signal with in-phase local oscillator, and the second baseband signal is generated by multiplying RF signal with 90° phase shifted local oscillator.

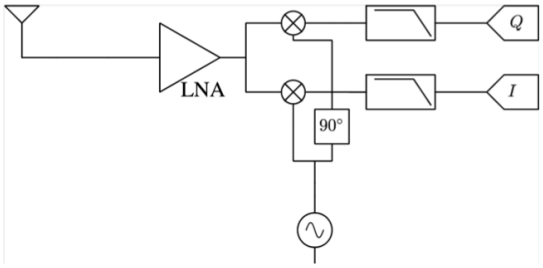


Fig. 2. Project Overview and Block Diagram

In a conventional communication system, receiver knows the modulation scheme of the transmitted wave and are cooperative, unlike in some applications such as cognitive radio and signal intelligence the modulation scheme of the baseband signal is not known [1], [2]. In such applications, modulation scheme of the received signal is determined using signal classification algorithms. A combination of properties of the baseband signal, such as amplitude, phase and frequency is varied depending on the sequence of symbols to encode information. Each type of variation is has its signature. If we inspect the Fig. 3(a), we can see in an amplitude modulation (AM-DSB) envelope of the transmitted wave is changing, for phase modulation we can observe that phase (BPSK, QPSK) of the signal is changing and for the frequency modulation (CPFSK) frequency of the signal is changing in time. Also as can be seen in Fig. 3(b) each modulation has its own spectrum signature. For the signal classification problem, such signatures in addition to other features can be used to

determine what kind of modulation scheme is used. These features can be crafted by experts, they can be extracted by Machine Learning (ML) algorithms or combination of these methods can be used [1].

Prior to ML and deep learning (DL) approaches, most of the works on signal classification focused on both hand crafted features and expert statistical modeling such as [3]–[6]. Expert modeling is costly and they may not be robust against the non-idealities presented in a real application such as multi-path propagation. DL based methods automatize this feature extraction process while taking account of non-idealities presented in a real application by taking advantage of the realistic data provided into the network. One of the earliest work where the advantages of DL based approach in signal classification is recognized is [2]. In [7], authors showed that a convolutional neural network (CNN) that is trained by synthetic data can successfully classify Over The Air transmission with high accuracy. Based on this, we will also use synthetic data extensively in our project to both train the models and to test them.

In this project we will compare two works. We will first inspect and analyze the performance of [2], where all of the important features of the signal modulation schemes are learned by the neural-network (NN) itself. Then, we will evaluate the performance of [8] using the implementation given in [9], where authors added a fixed linear combination layer after convolutional layer to model complex nature of the I/Q signals. We expect [8] to have better performance than [2] because, in [2] NN only learns one set of features for both I/Q input signals. Whereas [8] novel approach in handling I/Q signal also preserves the information between I and Q channels and account for relation between them [8].

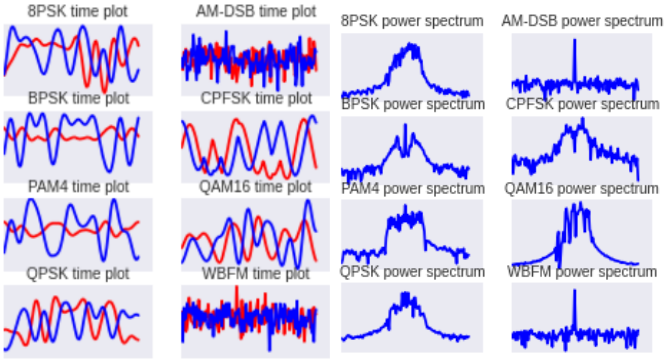
To compare all works fairly, we will train and test all of the works we mentioned by dataset given in [10]. Then we will generate additional synthetic test data using GNU radio, synthetic data generator developed by us and if time permits actual radio signal sources such as software defined radio and telecommunication test equipment. Also NN given in [2] contains 286,043 parameters, whereas proposed NN given in [8] contains 2,749,275 parameters. To account for performance increase due to increase in number of parameters, NN in [2] is scaled up to contain more parameters (2,707,547). We trained this scaled up NN when we compare the performance of [2] with [8].

II. Methods

A. CNN for Signal Classification

In [2], convolution of input (Z) with the extracted features (h) is fed into flattening and dense layers and output is one-hot encoded.

$$Z = \begin{array}{|c|c|}\hline I_1 & Q_1 \\ \hline I_2 & Q_2 \\ \hline I_3 & Q_3 \\ \hline \vdots & \vdots \\ \hline I_N & Q_N \\ \hline \end{array} \qquad h = \begin{array}{|c|}\hline h'_1 \\ \hline h'_2 \\ \hline h'_3 \\ \hline \vdots \\ \hline h'_M \\ \hline \end{array} \tag{1}$$



(a) Time Domain Modulation Examples [2]

(b) Power Spectral Density of Modulation Examples [2]

Fig. 3. Modulation Examples

If the input is 2xN and there are M features, we can represent the output of the convolutional layer as X_{DL}.

$$X_{DL} = Z \circledast h \tag{2}$$

$$X_{DL} = \begin{array}{|c|c|}\hline \uparrow & \uparrow \\ I \circledast h & Q \circledast h \\ \downarrow & \downarrow \\ \hline \end{array} \tag{3}$$

As we can observe in (3), we only extract one set of features for both I and Q channels and we weight these features in the following layers to guess the modulation method. Details of CNN model presented in [2] is given in Table-I and overview of the model can be seen in Fig.4.

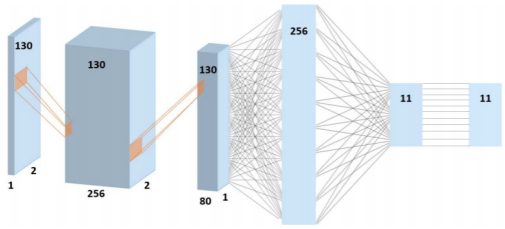


Fig. 4. CNN architecture for [2]

B. Complex CNN for Signal Classification

To take advantage of the relationship between I and Q signals, we would like to use a complex feature set. A complex feature set would have real and imaginary components and it would also enable us to model the complex nature of the I and Q signals. Unfortunately, current DL libraries only support real numbers. Therefore, to build a complex feature set we need to figure out how convolution of complex input with complex feature set would behave.

We denote complex feature set with h and each element of the feature set can be expressed as:

$$h_m = h'_m + jh''_m \tag{4}$$

Since our feature set compromises of M complex features, we can model our feature set as 2D variable as given in (5).

$$h = \begin{array}{|c|c|}\hline h'_1 & h''_1 \\ \hline h'_2 & h''_2 \\ \hline \vdots & \vdots \\ \hline h'_M & h''_M \\ \hline \end{array} \tag{5}$$

If we take complex convolution of input (Z) with our feature set (h):

$$Z \circledast h = (I + jQ) \circledast (h' + jh'') \tag{6}$$

$$= (I \circledast h' - Q \circledast h'') + j(I \circledast h'' + Q \circledast h') \tag{7}$$

In the output of our convolutional layer, we would like to have an expression that is similar to (7). But, if we take 2D convolution of the input (Z) with feature set (h) we would get:

$$X_{DL} = \begin{array}{|c|c|c|}\hline \uparrow & \uparrow & \uparrow \\ I \circledast h' & I \circledast h'' + Q \circledast h' & Q \circledast h'' \\ \downarrow & \downarrow & \downarrow \\ \hline \end{array} \tag{8}$$

Fortunately, complex convolution of Z and h is just a linear transformation of the 2D convolution of Z and h. If we apply the following linear transformation:

$$X = X_{DL} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{9}$$

We will get the desired result:

$$X = \begin{array}{|c|c|}\hline \uparrow & \uparrow \\ I \circledast h' - Q \circledast h'' & I \circledast h'' + Q \circledast h' \\ \downarrow & \downarrow \\ \hline \end{array} \tag{10}$$

In [8], to achieve complex convolution they follow the steps given above and add a linear transformation layer after the convolutional layer. After complex convolution is taken, it is fed into another convolutional layer, followed by flattening, dense and output layers. Details of this model is given in Table-II and NN architecture can be seen in Fig. 5.

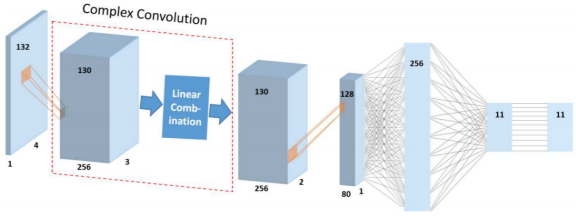


Fig. 5. CNN architecture for the complex model in [8]

C. Training

All the models use categorical cross entropy loss as the cost function and the stopping criteria is monitoring the validation loss. The model stops training when the validation cross entropy loss stops decreasing. It was observed that the models ran for at most 50 epochs before stopping. The optimization algorithm used was Adam. Adam is a popular stochastic gradient descent method which uses fist and second order moments.

No specific hyper-parameters were used while training the models. Overfitting was avoided by employing dropout layers in between the convolutional and densely connected layers.

The performance of the models were evaluated primarily by comparing the training and validation losses. In addition to this, the classification accuracy at different SNR levels was also taken into consideration. It was observed that classification accuracy is affected considerably by SNR. Training curves for [2] and [8] are given in Fig. 6 and Fig. 7 respectively.

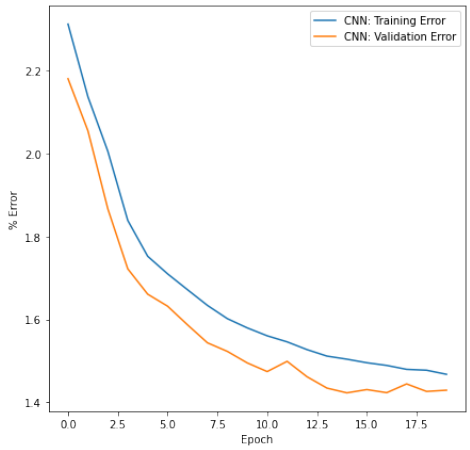


Fig. 6. Training and validation error for [2]

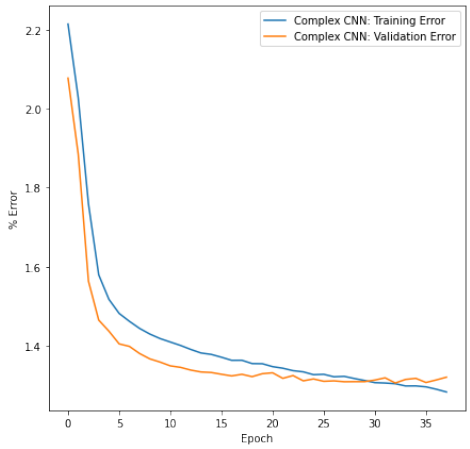


Fig. 7. Training and validation error for [8]

III. Implementation and Simulation

A. Simulation Procedure

The simulation procedure entails implementing the codes in [2] and [8] and observing the classifying accuracy across different Signal to Noise Ratio (SNR) levels. We have executed the code locally on our own machines.

TABLE I
MODEL DETAILS [2]

Layer Type	Input Size	Output Size	Details
Input	2 x 128	-	-
Zero Padding	2 x 128	2 x 130	Padding: 0, 0, 2
Convolution	2 x 130	256 x 2 x 130	Activation: ReLU, Kernel 1 x 3, Dropout: 0.5
Convolution	256 x 2 x 130	80 x 1 x 130	Activation: ReLU, Kernel: 2 x 1, Dropout: 0.5
Flatten	80 x 1 x 130	10400	-
Dense	10400	256	ReLU
Dense	256	11	Softmax
Output	11	11	One-hot encoded

TABLE II
MODEL DETAILS [8]

Layer Type	Input Size	Output Size	Details
Input	1 x 2 x 128	-	-
Zero Padding	1 x 2 x 128	1 x 4 x 132	Padding: 0, 2, 4
Convolution	1 x 4 x 132	256 x 3 x 130	Activation: None, Kernel: 2 x 3
Permute	256 x 3 x 130	256 x 130 x 3	-
Linear Transformation	256 x 130 x 3	256 x 130 x 2	-
Permute	256 x 130 x 2	256 x 2 x 130	-
Activation	256 x 2 x 130	256 x 2 x 130	Activation: ReLU
Dropout	256 x 2 x 130	256 x 2 x 130	Dropout = 0.5
Convolution	256 x 2 x 130	80 x 1 x 128	Activation: ReLU, Kernel: 2 x 3, Dropout = 0.5
Flatten	80 x 1 x 128	10240	-
Dense	10240	256	ReLU
Dense	256	11	ReLU
Softmax	11	11	One-Hot Output

B. Computation Platform

Our personal computes are being used as the computation platform. Presently this is a computer with an Intel i7 5770K, 64 GB of DDR4, 2 TB storage and two NVidia 1660 graphics cards with 6 GB of GDDR5 each. This computer is dual boated with Windows 10 and Ubuntu 20.04.

C. Software, Dependencies, and Environment

Python 3 primarily using the Anaconda Python distribution is the current environment for this project. The majority of the code is packaged in a install-able Python package complete with setup.py file. Currently Numpy, Scipy, Matplotlib, Keras, Scikit-Learn, PyTorch, tensorFlow, and Theano are the primary dependencies for the classification code. These can be installed using the Python or Anaconda package managers Pip and Conda. For data generation GNU Radio is being used for some modulation, impairments, and channel effects as it provides many of these abilities build in. Processing of captured data is done using GNU Radio and an Out Of Tree (OOT) package Gr Pdu Utils that aids in burst detection and segmentation of the data. GNU Radio and it's other Out Of Tree packages are build from source and installed on the Ubuntu partition. Collaboration on code is done using Git and a private repository on GitHub.

D. Data Collection

The original dataset used for the Army Signal Classification was limited access and is not longer available. Compatible datasets from other sources are available. The Army Signal Classification dataset is is described in enough detail to reproduce a similar dataset. DeepSig provides an alternative dataset with similar data for RF captures. These captures are a mix of simulated data and captured real signals using a software defined radio [10]. The reproduction of the dataset is being constructed by using tested implementations of both digital and analog modulations, channel effects, and radio impairments. Subset of the simulated dataset has impairments and channel effects applied and labeled to form a simulated data. This data makes it possible to directly control of impairment and channel parameters. Another subset of the modulated unimpaired is transmitted and received over a physical channel using a pair of SDR's to make the OTA dataset. Parameters radio impairments and channel effects are not directly controllable or knowable from this setup but are able to be estimated give knowledge of the original waveform. Because these two datasets are generated, the size of the dataset can be scaled to meet the needs of the project. Labeling of the captured dataset is done knowing what was transmitted and by hand. DeepSig dataset is a Python object serialization (pickle) file that consists of a dictionary which has keys that are a tuple that specifies modulation method and SNR. There are 11 different modulation methods in the dataset namely: 8PSK, AM-DSB, AM-SSB, BPSK, CPFSK, GFSK, PAM4, QAM16, QAM64, QPSK, WBFM with SNR levels ranging from -20dB to 18dB with 2dB increments. Each modulation, SNR pair in the dictionary consists of 1000 I/Q pairs with different impairments. Each I/Q pair consist of 128 entries. Training and testing data are randomly selected and half of the dataset is used for testing and other half is used for training. In summary dataset consists of 220,000 unique I/Q entries and

its size is 641MB. The shape of each sample is 128x2. No pre-processing was required before using the dataset apart from dividing it into training and test sets and one-hot mapping the desired outputs.

IV. Results

Using the code given in [9], we compared performance of [2] and [8]. We used half of the dataset given in [10] for training and other half for testings. Training and testing data selected randomly, same training and testing data are fed into the both algorithms. Fig. 10 shows our results and Fig. 11 shows the results of the original paper classification accuracy. The difference in the results is likely do to fewer number of training cycles we ran then the original paper.

Average accuracy on all SNR levels of [8] is around 50% and [2] is 48%. Accounting for relation between I/Q signals contributed 2% increase in accuracy in all SNR levels. Confusion matrix for [2] is given in Fig. 8 and confusion matrix for is given in Fig. 9. The results we replicated agree with results reported in [8]. At high SNR's the classification performance of the [8] out performs the [2] by about the average of 2 or 3 %. At SNR levels -8 to -2 dB the classification performance of [8] is as much as 10 % greater than [2]. This gives some indication that the complex network is better matched to the problem in a substantial way. A simple comparison is the performance of [8] is equivalent to [2] with an extra 2 dB of SNR while in the original paper this was not the case. This effect is likely caused by smaller number of training epochs used in this work.

The classification accuracy is found to be affected greatly by SNR levels as can be seen in Fig. 10. At SNR levels above 0 dB, the classification accuracy was seen to improve to about 80%, while the classification accuracy is as low as 10% at -20 dB SNR. The network seems to classify most of the signals as AM-SSB at lower SNR levels. This confusion of other modulations and AM-SSB at low SNR's is likely because of AM-SSB's lack from structure compared with other modulations and appear more like noise. At lower SNR's the dominant signal present is noise. For high SNR's most of the error comes from misclassification of similar modulations like QPSK and 8PSK or between QPSK, QAM16, and QAM64.

V. Discussion & Conclusion

Accounting for relation between I/Q signal resulted in slight improvement (2%) of the NN. We also expected an improvement in performance, but not by much because generally I and Q signals are highly correlated due to structure of the modulation methods, so introduction of mutual relation between them does not result in significant increase in information incorporated into the NN.

In this work, we demonstrated that, by including the relation between I/Q signal into network there is a small improvement in the performance for the modulation classification problem. We interpret this result as the fact that there is very little significant information in I/Q relation that would improve the accuracy of modulation classification by a great degree. This

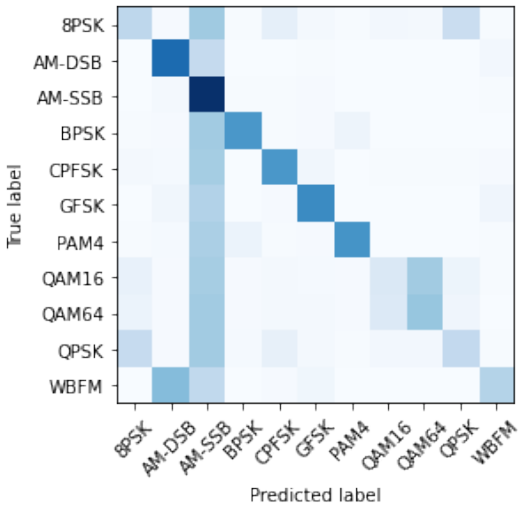


Fig. 8. Confusion Matrix for [2]

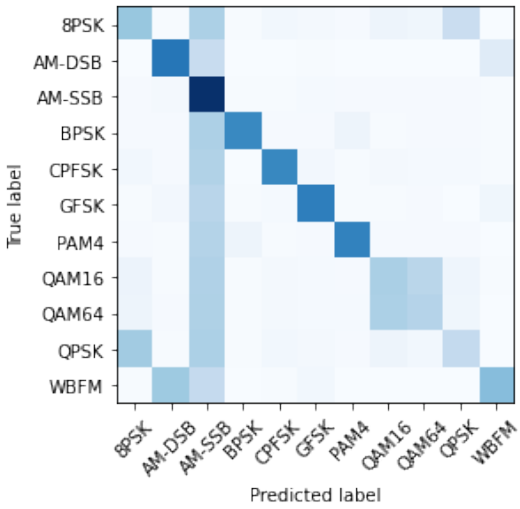


Fig. 9. Confusion Matrix for [8]

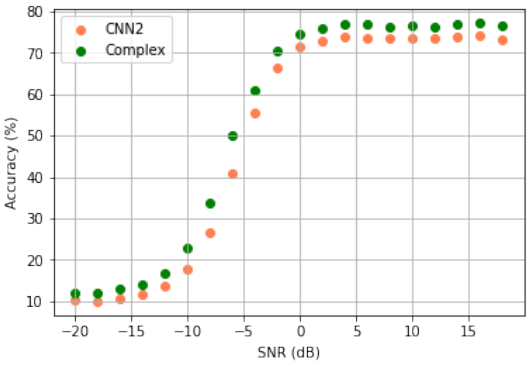


Fig. 10. Classification accuracy vs SNR

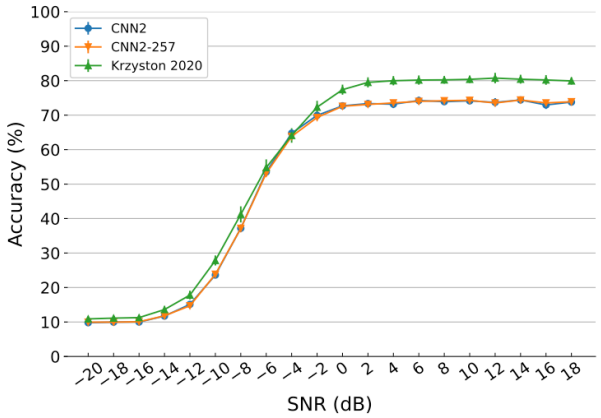


Fig. 11. Results from the original paper [8]

network can be used in various communication equipment where transmitter is not cooperative with the receiver so the receiver would need to decide which modulation method is used in the transmitted signal.

One limitation we observed is that there is limited preprocessing of data that went into the NN. In a future work the accuracy of the NN can be improved using some preprocessing before data goes into the NN, such as averaging, FFT to compute power spectrum and so on. An additional limitation is the fixed or corrected symbol rates present in the training data. This likely limits the currently trained model to the observations either of known or similar sample per symbol rates. Correcting for this can be done in future work by introducing variable rate resampling of the modulated data as a step in the training data generation.

Future works could improve upon the error of misclassification accuracy of similar modulations like what happens with QPSK, QAM16, and QAM64. In this case, the modulation QPSK is a valid subset of QAM16 and QAM64 with the same being true for QAM16 in QAM64. A number of methods possibly address this including increasing the data set to include more of these easily confused modulations, larger sample sizes, different scoring functions, or exploring alternatives to one hot encoding. Another related measure that could help with improving the classification of similar modulations would be to have the NN estimate the SNR for each modulation hypothesised. With the improvement in classification accuracy seen by applying complex convolution to the network's first layer, exploration into extending NN's to complex weights could possibly additionally improve performance and merits additional investigation. The data set contains perhaps the 11 most common modulations but no study of the robustness to modulations types that are not contained in the training set has been done. Additionally this network is dependent on prepossessing for the selection of samples containing a signal of interest. For positive SNR's this process can be done with a single energy detector however for lower SNR's this can be a non-trivial task. The addition of a noise class may prove helpful in this process of testing if any signal is present. This work shows the applicability of NN to the problem of RF signal classification.

VI. Disclaimer

A. Work Distribution

Each team member was be assigned with a particular study on the signal classification problem. Other works such as. literature review, report writing, data generation and collection, and running the programs, codes where distributed evenly between the team members. A brief summary of the future work distribution can be found in Table-III.

No one was harmed in the making of this work.

TABLE III
Work Distrubition

Assignee	Work Assigned	Percentage of Work
Wylie Standage-Beier	Report writing, packaging of the software, lit. review, generated additional synthetic data	$33.\overline{3}$
Muslum Emir Avci	Report writing, packaging of the software, run and evaluate [8]	$33.\overline{3}$
Muhammed Bilal	Report writing, Preprocessing data, Presentation video, run and evaluate [2]	$33.\overline{3}$

References

[1] K. Logue, E. Valles, A. Vila, A. Utter, D. Semmen, E. Grayver, S. Olsen, and D. Branchevsky, "Expert rf feature extraction to win the army rco ai signal classification challenge," *PROC. OF THE 18th PYTHON IN SCIENCE CONF. (SCIPY 2019)*, 2019.

[2] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in *International conference on engineering applications of neural networks*. Springer, 2016, pp. 213–226.

[3] S. S. Soliman and S.-Z. Hsue, "Signal classification using statistical moments," *IEEE Transactions on Communications*, vol. 40, no. 5, pp. 908–916, 1992.

[4] K. Kim, I. A. Akbar, K. K. Bae, J.-S. Um, C. M. Spooner, and J. H. Reed, "Cyclostationary approaches to signal detection and classification in cognitive radio," in *2007 2nd ieee international symposium on new frontiers in dynamic spectrum access networks*. IEEE, 2007, pp. 212–215.

[5] P. Isautier, J. Pan, R. DeSalvo, and S. E. Ralph, "Stokes space-based modulation format recognition for autonomous optical receivers," *Journal of lightwave technology*, vol. 33, no. 24, pp. 5157–5163, 2015.

[6] P. Isautier, J. Langston, J. Pan, and S. E. Ralph, "Agnostic software-defined coherent optical receiver performing time-domain hybrid modulation format recognition," in *Optical Fiber Communication Conference*. Optical Society of America, 2015, pp. Th2A–21.

[7] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[8] J. Krzyston, R. Bhattacharjea, and A. Stark, "Complex-valued convolutions for modulation recognition using deep learning," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2020, pp. 1–6.

[9] "Impementation of modulation pattern detection using complex convolutions in deep learning," https://github.com/JakobKrzyston/ Modulation_Pattern_Detection_Using_Complex_Convolutions, accessed: 2021-02-26.

[10] "Radioml dataset," https://github.com/radioML/dataset, accessed: 2021-02-26.

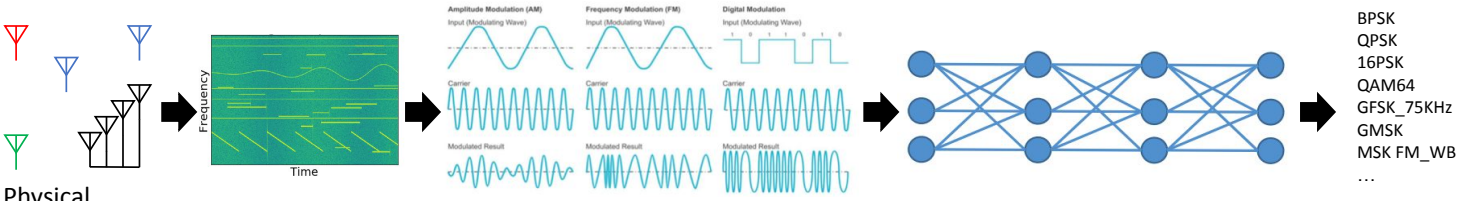
Page Intentionally Left Blank

Radio Signal Classification

EEE511 Spring 2021 Project
Team 7 Members:
Wylie Standage-Beier
Muslum Emir Avci
Muhammad Bilal

Keywords: —modulation, neural networks, machine learning, wireless communication, signals intelligence, cognitive radio

Radio Signal Classification



Physical Environment | Signal Capture | Recorded Signal | ML Method – Deep Learning | Classify

- Wireless communication systems modulate transmitted EM wave to encode information
- Conventional systems receivers use knowledge of the modulation to
  - Detection
  - Decode
- Absence of this knowledge the modulation must be inferred
  - Cognitive radio
  - Signal intelligence
- Signal classification algorithms are used for this inference

State of the Art

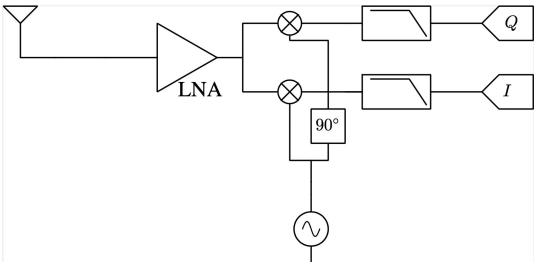
- 2018 Army Signal Classification Challenge
  - Put on by
    - The US Army's Rapid Capabilities Office
    - The MITRE Corporation
  - https://www.challenge.gov/challenge/army-signal-classification-challenge/

- Deep Learning Common to all winning teams
  - 1st place "Expert Feature Selection" and Deep Learning
  - 2nd and 3rd place employed only Deep Learning

- Data set
  - Custom data set
  - No longer available

Data Source and Structure

- Formatting data as complex time series
  - Different algebra, same information
  - Makes many operations easier
- Bandlimited
  - about baseband sample rate
  - Critically sampled
- Provided by DeepSig
  - https://github.com/radioML/dataset
  - Not the paper's dataset because it is no longer available



Receiver signal path

Antenna

Low noise amplifier

Mixer down conversion, splits into

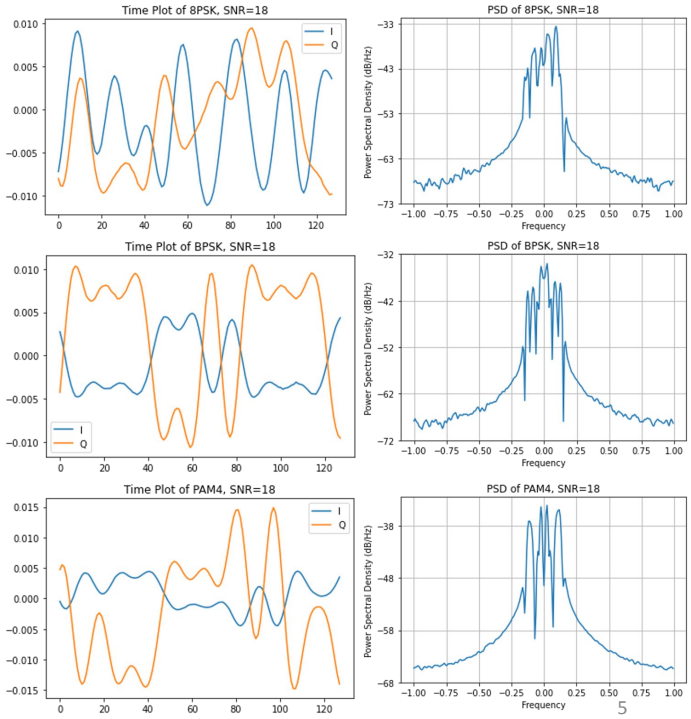
In-phase (I) and quadrature (Q) components

Low pass filter I and Q

Sample and digitize I and Q

Approaches and Background

- Classify signals to find modulation scheme
- Extract certain features are extracted from each modulation type
  - Changes in envelope
  - Phase jumps
- Automate feature extraction with Deep learning based methods
- Training deep learning on synthesized data
  - Achieves high accuracy with real data[1]



[1] - T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification,

Explanation of Dataset Used

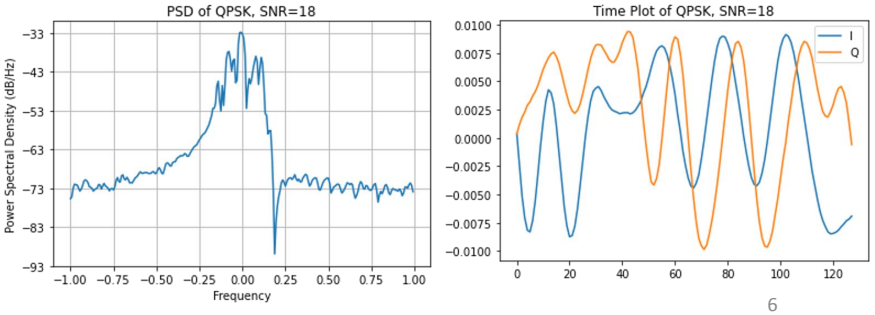
RADIOML 2016.10A BY DEEPSIG

- 11 modulation (8 digital + 3 analog)
- 110000 samples
- Shape of each sample: 2x128
- 128 I/Q pairs for each sample
- Size: 641 MB
- Mix of real and simulated data
- Dataset link: https://github.com/radioML/dataset

```
import pickle
import numpy as np

fh = open("<path to file>", 'rb')
ds = pickle.load(fh, encoding = 'latin1')

mod = 'QPSK'
snr = 18

# time signal
ts = ds[(mod, snr)][10, 0] + 1j * ds[(mod, snr)][10, 1]
```

Sample number — I

Sample number — Q



Setting up the Environment

- Anaconda Python Environment
  - https://www.anaconda.com/products/individual
- Installing packages using
  - Conda
  - Pip
- Hardware
  - CPU:
    - Intel i7 5770K
    - 64 GB of DDR4
    - 2 TB storage and
  - GPU:
    - two NVidia 1660's
    - 6 GB of GDDR5 each
- Providing code in Jupyter notebooks
  - Notebook for each method
  - Enables colocation of code and plots

Required Packages

- Numpy, Scipy, Matplotlib, Keras, Scikit-Learn, PyTorch, tensorFlow, and Theano
- Can be installed using Package Managers (PIP and Conda)

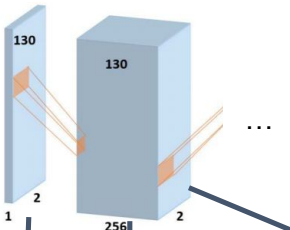
Code Sources

- **https://github.com/radioML/examples/blob/master/modulation_recognition/**
- **https://github.com/JakobKrzyston/Complex_Convolutions**

Approaches and Background

- Presenting two methods
  - CNN (2016)
    - Seminal work which introduced deep learning into this domain
    - Learns one set of features for I and Q
    - Does not account for relation between I and Q
  - Complex CNN (2020)
    - By handling complex numbers in a novel way, accuracy is increased
    - Accounts the relationship between I and Q channels by introducing a constant linear combination layer

CNN For Modulation Detection Input



In [1], NN only learns one set of features for both channels

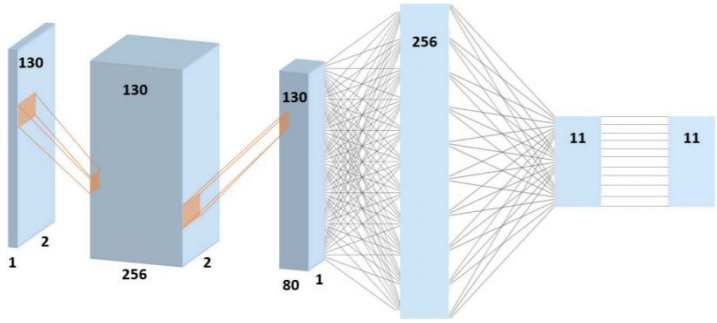
$$Z = \begin{bmatrix} I_1 & Q_1 \\ I_2 & Q_2 \\ I_3 & Q_3 \\ \vdots & \vdots \\ I_N & Q_N \end{bmatrix} \quad h = \begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \\ \vdots \\ h'_M \end{bmatrix} \quad X_{\mathrm{DL}} = \begin{bmatrix} I_1 & Q_1 \\ I_2 & Q_2 \\ I_3 & Q_3 \\ \vdots & \vdots \\ I_N & Q_N \end{bmatrix} \circledast \begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \\ \vdots \\ h'_M \end{bmatrix} \quad X_{\mathrm{DL}} = \begin{bmatrix} \uparrow & \uparrow \\ I \circledast h' & Q \circledast h' \\ \downarrow & \downarrow \end{bmatrix}$$

[1] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in International conference on engineering applications of neural networks. Springer, 2016, pp. 213–226.

CNN For Modulation Detection NN Structure [1]

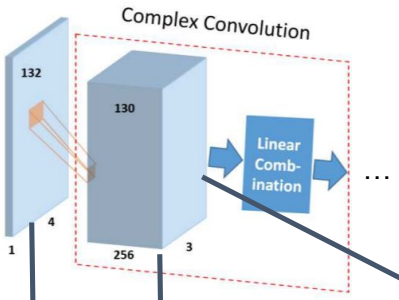
- Classifying modulation using Convolutional Neural Network
- Basic layers
  - Input/Output
  - Convolutional layers
  - Dense layers
  - Dropout
  - Padding
  - Flatten layers

Layer Type	Input Size	Output Size	Details
Input	2 x 128	-	-
Zero Padding	2 x 128	2 x 130	Padding: 0, 0, 2
Convolution	2 x 130	256 x 2 x 130	Activation: ReLU, Kernel 1 x 3, Dropout: 0.5
Convolution	256 x 2 x 130	80 x 1 x 130	Activation: ReLU, Kernel: 2 x 1, Dropout: 0.5
Flatten	80 x 1 x 130	10400	-
Dense	10400	256	ReLU
Dense	256	11	Softmax
Output	11	11	One-hot encoded



[1] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in International conference on engineering applications of neural networks. Springer, 2016, pp. 213–226.

Complex CNN For Modulation Detection Input



In [2], NN effectively learns a complex feature set for accounting the relationship between I and Q

$$h_m = h'_m + jh''_m$$

$$X = (I+jQ)\circledast(h'+jh'') = (I\circledast h' - Q\circledast h'') + j(I\circledast h'' + Q\circledast h')$$

$$X_{\mathrm{DL}} = \begin{array}{|c|c|c|} \hline \uparrow & \uparrow & \uparrow \\ I\circledast h' & I\circledast h'' + Q\circledast h' & Q\circledast h'' \\ \downarrow & \downarrow & \downarrow \\ \hline \end{array}$$

$$Z = \begin{array}{|c|c|} \hline I_1 & Q_1 \\ I_2 & Q_2 \\ I_3 & Q_3 \\ \vdots & \vdots \\ I_N & Q_N \\ \hline \end{array} \quad h = \begin{array}{|c|c|} \hline h'_1 & h''_1 \\ h'_2 & h''_2 \\ h'_3 & h''_3 \\ \vdots & \vdots \\ h'_M & h''_M \\ \hline \end{array} \quad X_{\mathrm{DL}} = \begin{array}{|c|c|} \hline I_1 & Q_1 \\ I_2 & Q_2 \\ I_3 & Q_3 \\ \vdots & \vdots \\ I_N & Q_N \\ \hline \end{array} \circledast \begin{array}{|c|c|} \hline h'_1 & h''_1 \\ h'_2 & h''_2 \\ h'_3 & h''_3 \\ \vdots & \vdots \\ h'_M & h''_M \\ \hline \end{array}$$

$$X = X_{\mathrm{DL}} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \end{bmatrix}$$

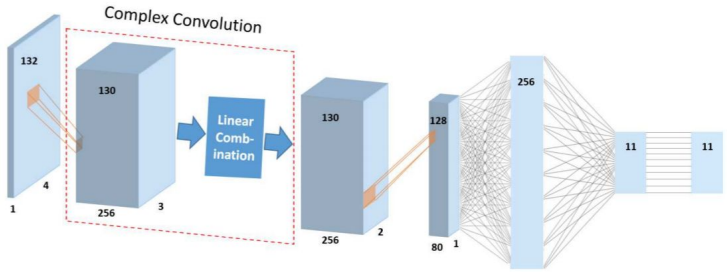
$$X = \begin{array}{|c|c|} \hline \uparrow & \uparrow \\ I\circledast h' - Q\circledast h'' & I\circledast h'' + Q\circledast h' \\ \downarrow & \downarrow \\ \hline \end{array}$$

[2] J. Krzyston, R. Bhattacharjea, and A. Stark, "Complex-valued convolutions for modulation recognition using deep learning," in 2020 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, 2020, pp. 1–6.

Complex CNN For Modulation Detection NN Structure [2]

- Classifying modulation using Convolutional Neural Network
  - Complex valued weights
- Basic layers
  - Input/Output
  - Convolutional layers
  - Layer for linear combination
  - Dense layers
  - Dropout
  - Padding
  - Flatten layers

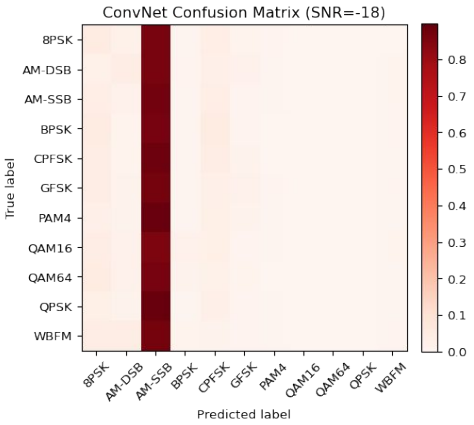
Layer Type	Input Size	Output Size	Details
Input	1 x 2 x 128	-	-
Zero Padding	1 x 2 x 128	1 x 4 x 132	Padding: 0, 2, 4
Convolution	1 x 4 x 132	256 x 3 x 130	Activation: None, Kernel: 2 x 3
Permute	256 x 3 x 130	256 x 130 x 3	-
Linear Transformation	256 x 130 x 3	256 x 130 x 2	-
Permute	256 x 130 x 2	256 x 2 x 130	-
Activation	256 x 2 x 130	256 x 2 x 130	Activation: ReLU
Dropout	256 x 2 x 130	256 x 2 x 130	Dropout = 0.5
Convolution	256 x 2 x 130	80 x 1 x 128	Activation: ReLU, Kernel: 2 x 3, Dropout = 0.5
Flatten	80 x 1 x 128	10240	-
Dense	10240	256	ReLU
Dense	256	11	ReLU
Softmax	11	11	One-Hot Output



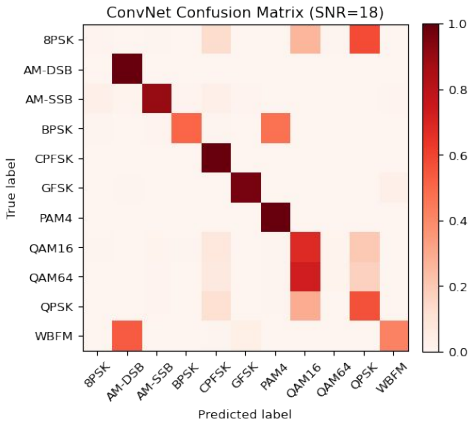
[2] J. Krzyston, R. Bhattacharjea, and A. Stark, "Complex-valued convolutions for modulation recognition using deep learning," in 2020 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, 2020, pp. 1–6.

Results/Comparison

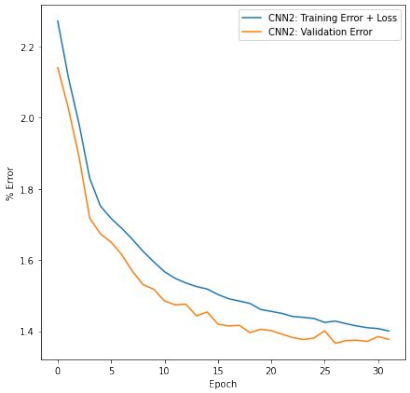
Comparison of CNN performance at different SNR Levels



At SNR = -18 dB

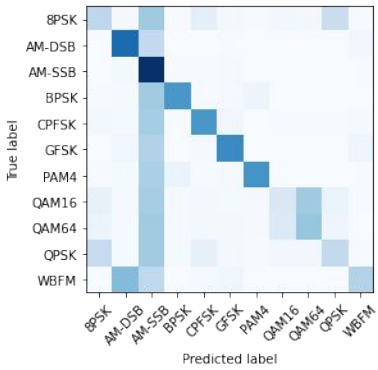


At SNR = 18 dB



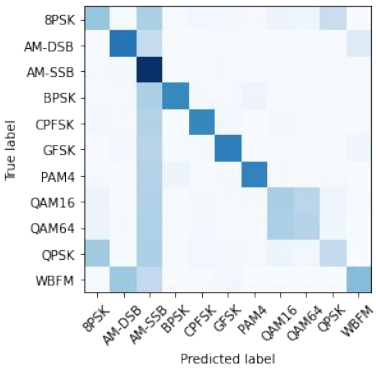
Training and Validation Error

Results/Comparison



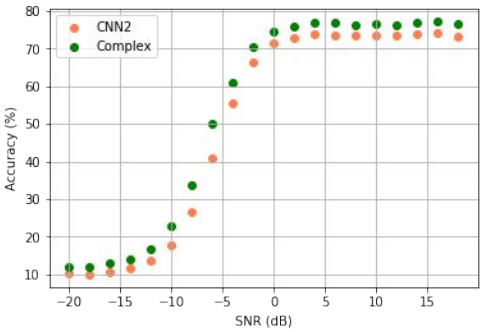
CNN Confusion Matrix

Overall Accuracy



Complex CNN Confusion Matrix

Overall Accuracy

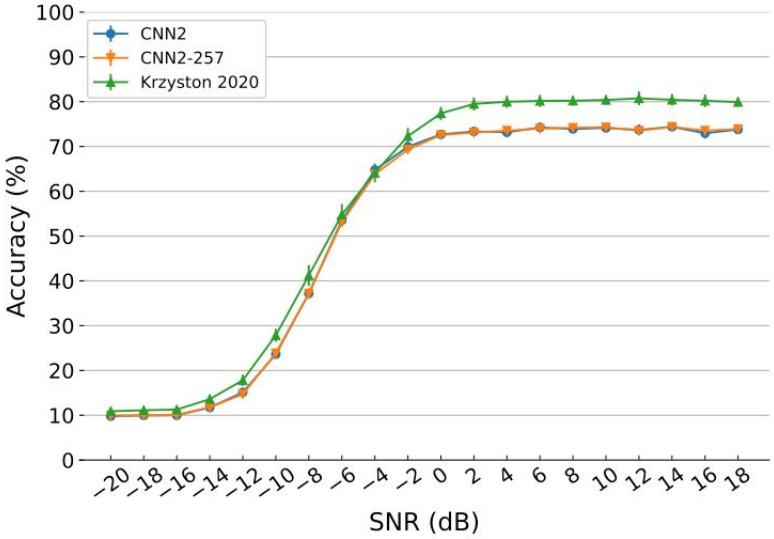


Comparison of CNN and Complex CNN for different SNR Levels

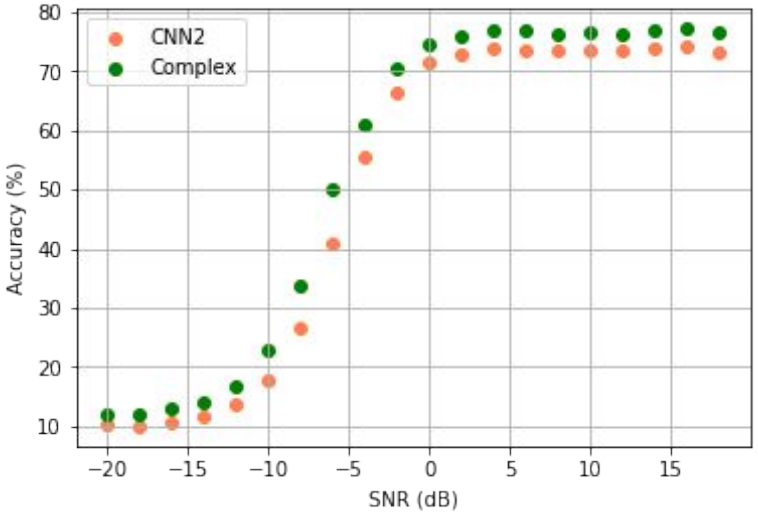
Accounting for relation between I/Q signal resulted in slight improvement (2%) of the neural network.

Comparison to the original paper

Original Paper



Replicated Results



Conclusion

- Classification of communication signals using deep learning
  - high degree of accuracy achieved
  - Still dependent on SNR
  - Misclassifies modulations which are inherently similar
    - Leads to lower overall accuracy

- Accounting for data structure improves accuracy
  - Operating on complex IQ over real pairs I and Q
  - Learning complex numbers improves accuracy

- Training on synthetic data
  - Transferible to real data

Radio Signal Classification

EEE511 Spring 2021 Project

Team 7 Members:
Wylie Standage-Beier
Muslum Emir Avci
Muhammad Bilal

Keywords: —modulation, neural networks, machine learning, wireless communication, signals intelligence

Page Intentionally Left Blank

Final Code

April 24, 2021

[1]:
```
#importing the required packages
import os,random
import numpy as np
os.environ["KERAS_BACKEND"] = "tensorflow"
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
from keras.utils import np_utils
import keras.models as models
from keras.layers.core import␣
 ↪Reshape,Dense,Dropout,Activation,Flatten,Lambda,Permute
from keras.layers.noise import GaussianNoise
from keras.layers.convolutional import Convolution2D, MaxPooling2D,␣
 ↪ZeroPadding2D
from keras.regularizers import *
from keras.optimizers import Adam
import matplotlib.pyplot as plt
%matplotlib inline
import keras
import pickle, random, time
import _pickle as cPickle
```

[2]:
```
#checking if gpu is available
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.experimental.
 ↪list_physical_devices('GPU')))
tf.config.list_physical_devices('GPU')
```

```
Num GPUs Available:  1
```

[2]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

[3]:
```
#loading the dataset into memory
Xd = pickle.load(open("RML2016.10a_dict.pkl", 'rb'), encoding = 'latin1')
test_snrs,mods = map(lambda j: sorted( list( set( map( lambda x: x[j], Xd.
 ↪keys() ) ) ) ), [1,0])
X = []
lbl = []

for mod in mods:
```

```
    for snr in test_snrs:
        X.append(Xd[(mod,snr)])
        for i in range(Xd[(mod,snr)].shape[0]):  lbl.append((mod,snr))
X = np.vstack(X)
print(X.shape)
```

```
(220000, 2, 128)
```

[4]:
```
#partitioning the data into test and train data and one-hot encoding the truth␣
 ↪values
np.random.seed(2019)
n_examples = X.shape[0]
n_train    = int(round(n_examples * 0.5))
train_idx  = np.random.choice(range(0,n_examples), size=n_train, replace=False)
test_idx   = list(set(range(0,n_examples))-set(train_idx))
X_train    = X[train_idx]
X_test     = X[test_idx]

#function to one-hot encode the truth values
def to_onehot(yy):
    yy1 = np.zeros([len(yy) ,max(yy)+1])
    yy1[  np.arange(len(yy)),yy] = 1 # ?
    return yy1
Y_train = to_onehot(list(map(lambda x: mods.index(lbl[x][0]), train_idx)))
Y_test = to_onehot(list(map(lambda x: mods.index(lbl[x][0]), test_idx)))

in_shp = list(X_train.shape[1:])
print(X_train.shape, in_shp)
classes = mods
```

```
(110000, 2, 128) [2, 128]
```

[5]:
```
#building the CNN model explained in the first paper
dr = 0.5 #dropout rate (50%)
cnn = models.Sequential()
cnn.add(Reshape([1]+in_shp, input_shape=in_shp))
cnn.add(ZeroPadding2D((0, 2),data_format='channels_first'))
cnn.add(Convolution2D(256, (1, 3), padding='valid', activation="relu",␣
 ↪name="conv1", kernel_initializer='glorot_uniform',␣
 ↪data_format='channels_first'))#ch from 3->4
cnn.add(Dropout(dr))
cnn.add(Convolution2D(80, (2, 1), padding='valid', activation="relu",␣
 ↪name="conv2", kernel_initializer='glorot_uniform',␣
 ↪data_format='channels_first'))
cnn.add(Dropout(dr))
cnn.add(Flatten())
```

```
cnn.add(Dense(256, activation='relu', kernel_initializer='he_normal',␣
 ↪name="dense1"))
cnn.add(Dropout(dr))
cnn.add(Dense(len(classes), kernel_initializer='he_normal', name="dense2"))
cnn.add(Activation('softmax'))
cnn.add(Reshape([len(classes)]))
cnn.compile(loss='categorical_crossentropy', optimizer='adam')
cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape (Reshape)            (None, 1, 2, 128)         0
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 1, 2, 132)         0
_________________________________________________________________
conv1 (Conv2D)               (None, 256, 2, 130)       1024
_________________________________________________________________
dropout (Dropout)            (None, 256, 2, 130)       0
_________________________________________________________________
conv2 (Conv2D)               (None, 80, 1, 130)        41040
_________________________________________________________________
dropout_1 (Dropout)          (None, 80, 1, 130)        0
_________________________________________________________________
flatten (Flatten)            (None, 10400)             0
_________________________________________________________________
dense1 (Dense)               (None, 256)               2662656
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense2 (Dense)               (None, 11)                2827
_________________________________________________________________
activation (Activation)      (None, 11)                0
_________________________________________________________________
reshape_1 (Reshape)          (None, 11)                0
=================================================================
Total params: 2,707,547
Trainable params: 2,707,547
Non-trainable params: 0
_________________________________________________________________
```

[6]:
```
#Defining the linear combination function explained in the second paper
def LC(x):
    import keras.backend as K
    y = K.constant([0, 1, 0, -1, 0, 1],shape=[2,3])
    return K.dot(x,K.transpose(y))
```

[7]:
```
#Building the complex CNN defined in the second paper
complex_CNN = models.Sequential()
complex_CNN.add(Reshape([1]+in_shp, input_shape=in_shp))
complex_CNN.add(ZeroPadding2D((1, 2),data_format='channels_first'))
complex_CNN.add(Convolution2D(256, (2, 3), padding='valid',␣
 ↪activation='linear', name="conv1", kernel_initializer='glorot_uniform',␣
 ↪data_format='channels_first'))#ch from 3->4
complex_CNN.add(Permute((1,3,2)))
complex_CNN.add(Lambda(LC))
complex_CNN.add(Permute((1,3,2)))
complex_CNN.add(Activation('relu'))
complex_CNN.add(Dropout(dr))
complex_CNN.add(Convolution2D(80, (2, 3), padding='valid', activation="relu",␣
 ↪name="conv2", kernel_initializer='glorot_uniform',␣
 ↪data_format='channels_first'))
complex_CNN.add(Dropout(dr))
complex_CNN.add(Flatten())
complex_CNN.add(Dense(256, activation='relu', kernel_initializer='he_normal',␣
 ↪name="dense1"))
complex_CNN.add(Dropout(dr))
complex_CNN.add(Dense( len(classes), kernel_initializer='he_normal',␣
 ↪name="dense2" ))
complex_CNN.add(Activation('softmax'))
complex_CNN.add(Reshape([len(classes)]))
complex_CNN.compile(loss='categorical_crossentropy', optimizer='adam')
complex_CNN.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_2 (Reshape)          (None, 1, 2, 128)         0
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 1, 4, 132)         0
_________________________________________________________________
conv1 (Conv2D)               (None, 256, 3, 130)       1792
_________________________________________________________________
permute (Permute)            (None, 256, 130, 3)       0
_________________________________________________________________
lambda (Lambda)              (None, 256, 130, 2)       0
_________________________________________________________________
permute_1 (Permute)          (None, 256, 2, 130)       0
_________________________________________________________________
activation_1 (Activation)    (None, 256, 2, 130)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 256, 2, 130)       0
_________________________________________________________________
```

```
conv2 (Conv2D)               (None, 80, 1, 128)        122960
_________________________________________________________________
dropout_4 (Dropout)          (None, 80, 1, 128)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 10240)             0
_________________________________________________________________
dense1 (Dense)               (None, 256)               2621696
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense2 (Dense)               (None, 11)                2827
_________________________________________________________________
activation_2 (Activation)    (None, 11)                0
_________________________________________________________________
reshape_3 (Reshape)          (None, 11)                0
=================================================================
Total params: 2,749,275
Trainable params: 2,749,275
Non-trainable params: 0
_________________________________________________________________
```

[8]:
```
# Number of epochs
epochs = 100
# Training batch size
batch_size = 1024
```

[9]:
```
#Training the first CNN model
start = time.time()
filepath = 'cnn.wts.h5'
history_cnn = cnn.fit(X_train, Y_train, batch_size=batch_size,
    epochs=epochs,
    verbose=2,
    validation_data=(X_test, Y_test),
    callbacks = [
        keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss',␣
 ↪verbose=0, save_best_only=True, mode='auto'),
        keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,␣
 ↪verbose=0, mode='auto')
    ])
cnn.load_weights(filepath)
end = time.time()
duration = end - start
print('CNN Training time = ' + str(round(duration/60,5)) + 'minutes')

#Training the second complex CNN model
start = time.time()
filepath = 'complex.wts.h5'
history_complex = complex_CNN.fit(X_train,
```

```
    Y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=2,
    validation_data=(X_test, Y_test),
    callbacks = [
        keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss',␣
 ↪verbose=0, save_best_only=True, mode='auto'),
        keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,␣
 ↪verbose=0, mode='auto')
    ])
complex_CNN.load_weights(filepath)
end = time.time()
duration = end - start
print('Complex Training time = ' + str(round(duration/60,5)) + 'minutes')
```

```
Epoch 1/100
108/108 - 12s - loss: 2.3112 - val_loss: 2.1797
Epoch 2/100
108/108 - 4s - loss: 2.1362 - val_loss: 2.0542
Epoch 3/100
108/108 - 4s - loss: 2.0047 - val_loss: 1.8661
Epoch 4/100
108/108 - 5s - loss: 1.8383 - val_loss: 1.7217
Epoch 5/100
108/108 - 5s - loss: 1.7520 - val_loss: 1.6607
Epoch 6/100
108/108 - 5s - loss: 1.7096 - val_loss: 1.6317
Epoch 7/100
108/108 - 5s - loss: 1.6714 - val_loss: 1.5867
Epoch 8/100
108/108 - 5s - loss: 1.6336 - val_loss: 1.5435
Epoch 9/100
108/108 - 5s - loss: 1.6016 - val_loss: 1.5228
Epoch 10/100
108/108 - 5s - loss: 1.5794 - val_loss: 1.4950
Epoch 11/100
108/108 - 5s - loss: 1.5603 - val_loss: 1.4742
Epoch 12/100
108/108 - 5s - loss: 1.5458 - val_loss: 1.4989
Epoch 13/100
108/108 - 5s - loss: 1.5270 - val_loss: 1.4617
Epoch 14/100
108/108 - 5s - loss: 1.5115 - val_loss: 1.4344
Epoch 15/100
108/108 - 5s - loss: 1.5041 - val_loss: 1.4229
Epoch 16/100
```

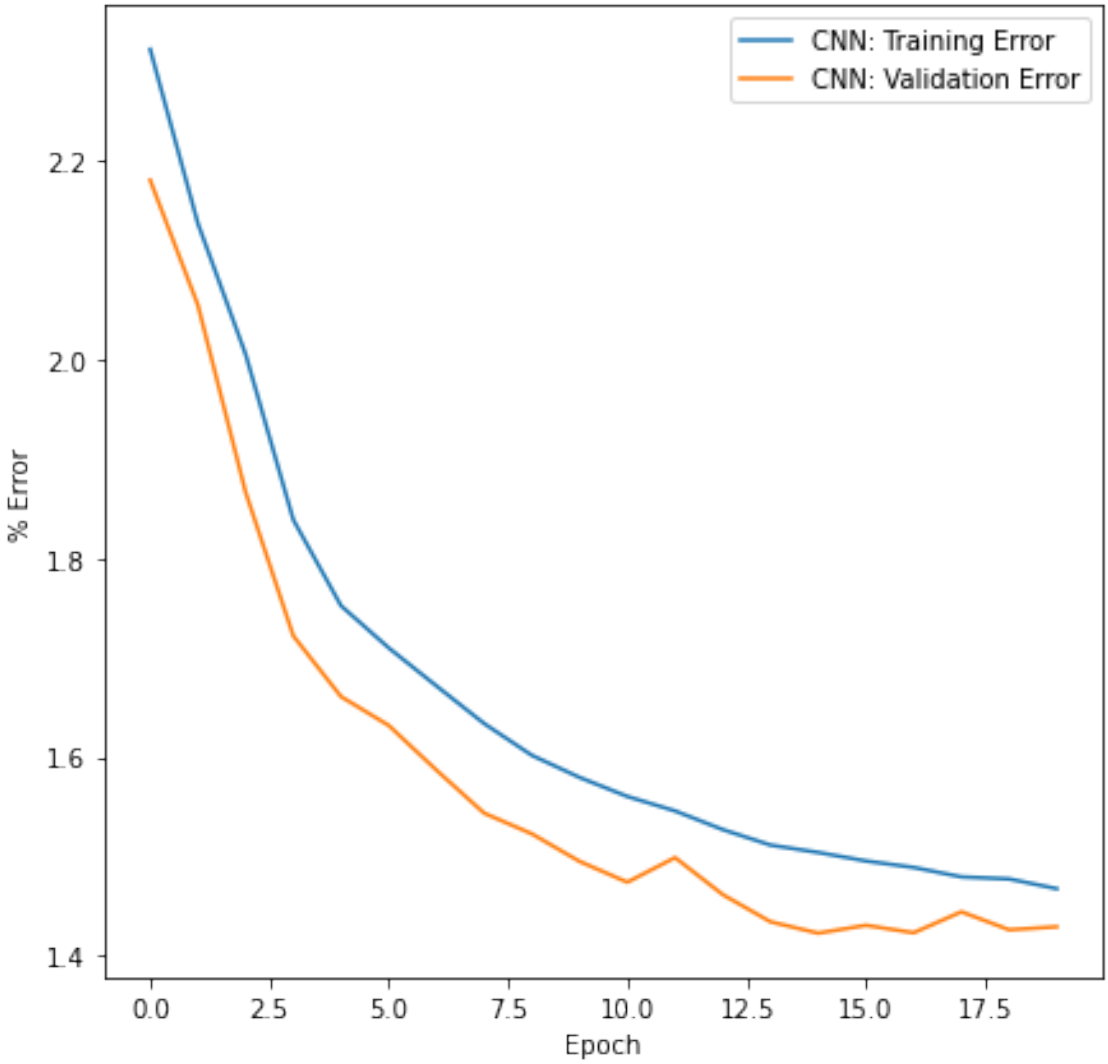
```
108/108 - 5s - loss: 1.4955 - val_loss: 1.4308
Epoch 17/100
108/108 - 5s - loss: 1.4890 - val_loss: 1.4233
Epoch 18/100
108/108 - 5s - loss: 1.4795 - val_loss: 1.4444
Epoch 19/100
108/108 - 5s - loss: 1.4774 - val_loss: 1.4264
Epoch 20/100
108/108 - 5s - loss: 1.4676 - val_loss: 1.4293
CNN Training time = 1.64672minutes
Epoch 1/100
108/108 - 13s - loss: 2.2141 - val_loss: 2.0773
Epoch 2/100
108/108 - 11s - loss: 2.0271 - val_loss: 1.8844
Epoch 3/100
108/108 - 11s - loss: 1.7591 - val_loss: 1.5636
Epoch 4/100
108/108 - 11s - loss: 1.5798 - val_loss: 1.4652
Epoch 5/100
108/108 - 11s - loss: 1.5176 - val_loss: 1.4371
Epoch 6/100
108/108 - 11s - loss: 1.4820 - val_loss: 1.4050
Epoch 7/100
108/108 - 11s - loss: 1.4621 - val_loss: 1.3986
Epoch 8/100
108/108 - 11s - loss: 1.4441 - val_loss: 1.3806
Epoch 9/100
108/108 - 11s - loss: 1.4298 - val_loss: 1.3670
Epoch 10/100
108/108 - 11s - loss: 1.4185 - val_loss: 1.3589
Epoch 11/100
108/108 - 11s - loss: 1.4097 - val_loss: 1.3494
Epoch 12/100
108/108 - 11s - loss: 1.4010 - val_loss: 1.3456
Epoch 13/100
108/108 - 11s - loss: 1.3909 - val_loss: 1.3387
Epoch 14/100
108/108 - 11s - loss: 1.3820 - val_loss: 1.3340
Epoch 15/100
108/108 - 11s - loss: 1.3784 - val_loss: 1.3326
Epoch 16/100
108/108 - 11s - loss: 1.3712 - val_loss: 1.3281
Epoch 17/100
108/108 - 11s - loss: 1.3634 - val_loss: 1.3239
Epoch 18/100
108/108 - 11s - loss: 1.3635 - val_loss: 1.3280
Epoch 19/100
108/108 - 11s - loss: 1.3551 - val_loss: 1.3221
```

```
Epoch 20/100
108/108 - 11s - loss: 1.3548 - val_loss: 1.3296
Epoch 21/100
108/108 - 11s - loss: 1.3474 - val_loss: 1.3319
Epoch 22/100
108/108 - 11s - loss: 1.3434 - val_loss: 1.3178
Epoch 23/100
108/108 - 11s - loss: 1.3373 - val_loss: 1.3248
Epoch 24/100
108/108 - 11s - loss: 1.3343 - val_loss: 1.3115
Epoch 25/100
108/108 - 11s - loss: 1.3271 - val_loss: 1.3162
Epoch 26/100
108/108 - 11s - loss: 1.3280 - val_loss: 1.3103
Epoch 27/100
108/108 - 11s - loss: 1.3217 - val_loss: 1.3114
Epoch 28/100
108/108 - 11s - loss: 1.3229 - val_loss: 1.3092
Epoch 29/100
108/108 - 11s - loss: 1.3173 - val_loss: 1.3094
Epoch 30/100
108/108 - 11s - loss: 1.3123 - val_loss: 1.3094
Epoch 31/100
108/108 - 11s - loss: 1.3069 - val_loss: 1.3134
Epoch 32/100
108/108 - 11s - loss: 1.3060 - val_loss: 1.3191
Epoch 33/100
108/108 - 11s - loss: 1.3041 - val_loss: 1.3060
Epoch 34/100
108/108 - 11s - loss: 1.2985 - val_loss: 1.3150
Epoch 35/100
108/108 - 11s - loss: 1.2987 - val_loss: 1.3178
Epoch 36/100
108/108 - 11s - loss: 1.2963 - val_loss: 1.3073
Epoch 37/100
108/108 - 11s - loss: 1.2903 - val_loss: 1.3137
Epoch 38/100
108/108 - 11s - loss: 1.2833 - val_loss: 1.3207
Complex Training time = 7.16287minutes
```

[10]:
```
#Plotting the training and validation error for the first CNN
plt.figure(figsize = (7,7))
plt.plot(history_cnn.epoch, history_cnn.history['loss'], label="CNN: Training␣
 ↪Error")
plt.plot(history_cnn.epoch, history_cnn.history['val_loss'], label="CNN:␣
 ↪Validation Error")
plt.xlabel('Epoch')
```

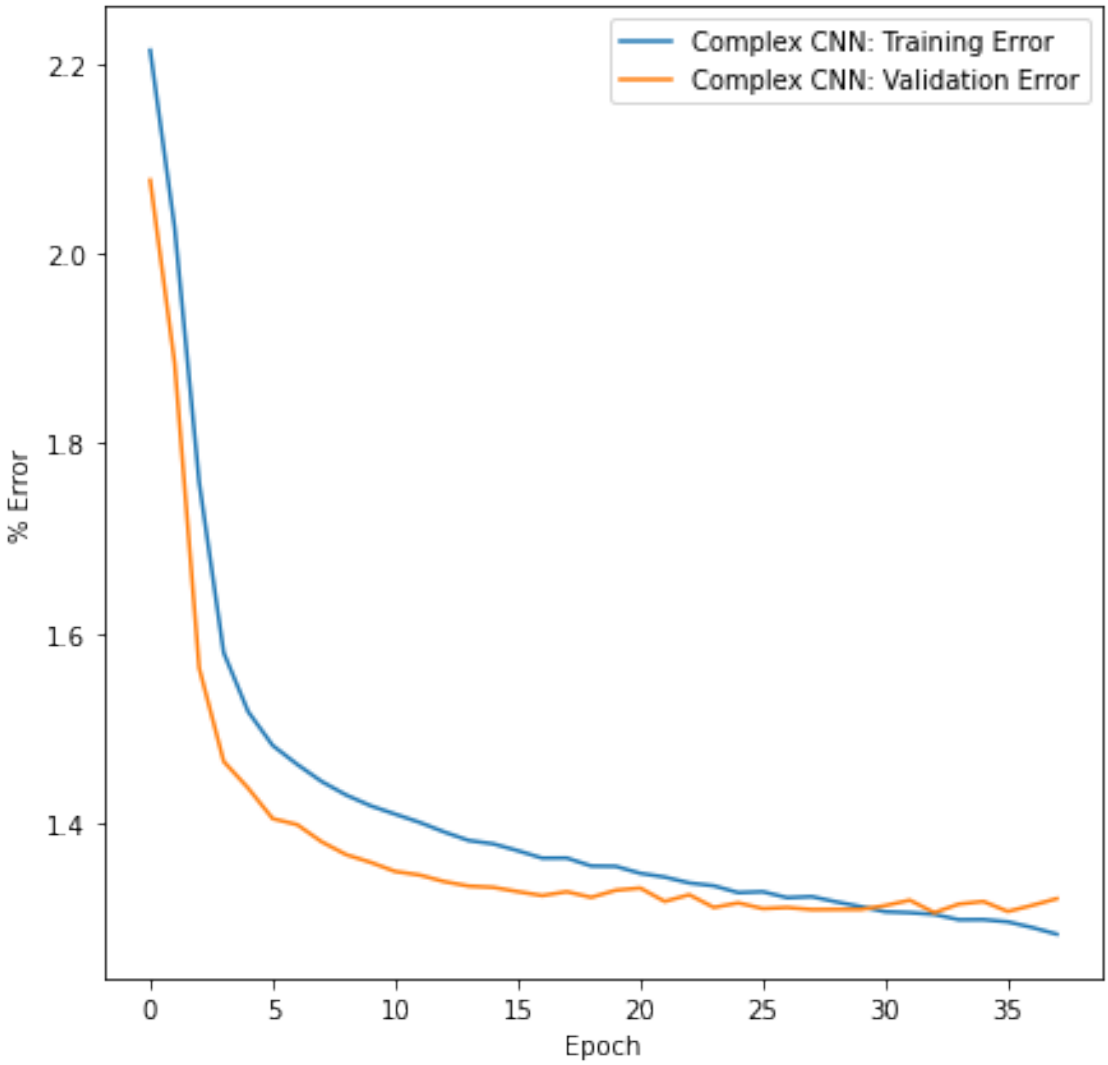
```
plt.ylabel('% Error')
plt.legend()
```

[10]: `<matplotlib.legend.Legend at 0x26ac4903eb0>`



[21]:
```
#Plotting the training and validation error for the second complex CNN
plt.figure(figsize = (7,7))
plt.plot(history_complex.epoch, history_complex.history['loss'], label='Complex␣
 ↪CNN: Training Error')
plt.plot(history_complex.epoch, history_complex.history['val_loss'],␣
 ↪label='Complex CNN: Validation Error')
plt.xlabel('Epoch')
plt.ylabel('% Error')
plt.legend()
```

[21]: `<matplotlib.legend.Legend at 0x26a8c92c7f0>`

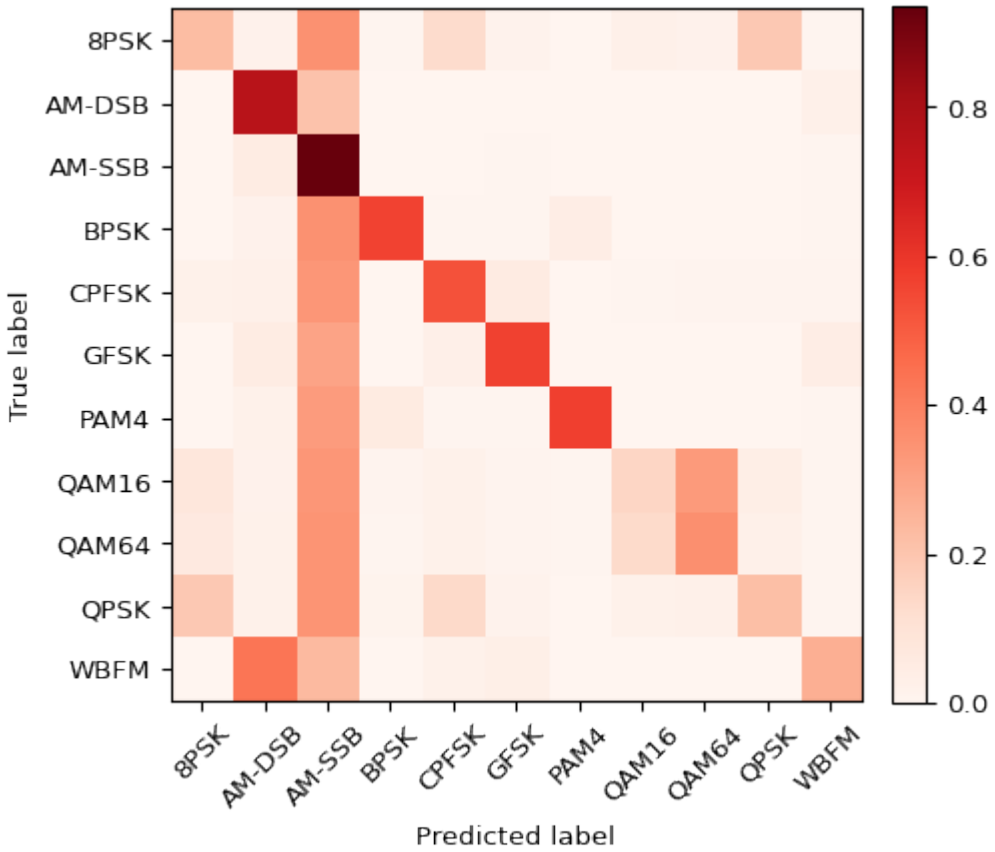


[12]:
```
#Creating a function to plot the confusion matrices
def plot_confusion_matrix(cm, title='', cmap=plt.cm.Reds, labels=[]):
    my_dpi=96
    plt.figure(figsize=(500/my_dpi, 500/my_dpi), dpi=my_dpi)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar(fraction=0.046, pad=0.04)
    tick_marks = np.arange(len(labels))
    plt.xticks(tick_marks, labels, rotation=45)
    plt.yticks(tick_marks, labels)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

[13]:
```
#Plotting the confusion matrix for the first CNN
test_Y_hat = cnn.predict(X_test, batch_size=batch_size)
conf = np.zeros([len(classes),len(classes)])
confnorm = np.zeros([len(classes),len(classes)])
for i in range(0,X_test.shape[0]):
    j = list(Y_test[i,:]).index(1)
    k = int(np.argmax(test_Y_hat[i,:]))
    conf[j,k] = conf[j,k] + 1
for i in range(0,len(classes)):
    confnorm[i,:] = conf[i,:] / np.sum(conf[i,:])
cor = np.sum(np.diag(conf))
ncor = np.sum(conf) - cor
print("Overall Accuracy - CNN: ", cor / (cor+ncor))
acc = 1.0*cor/(cor+ncor)
plt.figure()
plot_confusion_matrix(confnorm, labels=classes)
```

```
Overall Accuracy - CNN:  0.4678
```

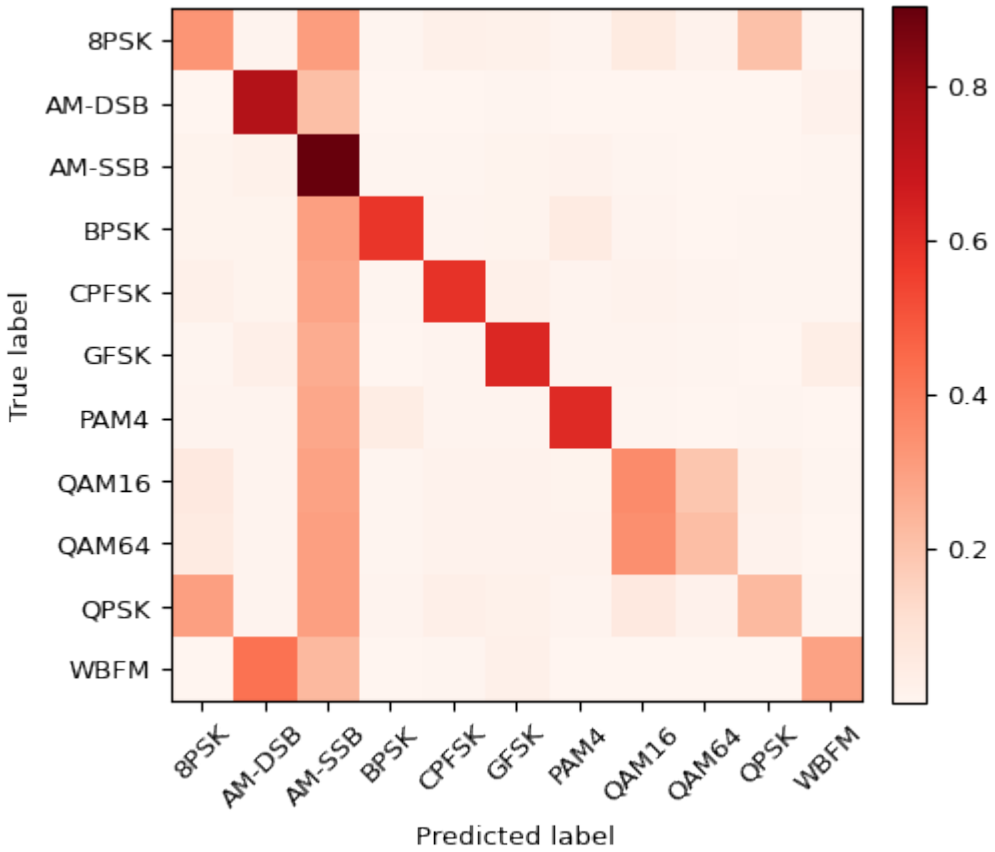
```
<Figure size 432x288 with 0 Axes>
```



[14]:
```
#Plotting the confusion matrix for the second complex CNN
test_Y_hat = complex_CNN.predict(X_test, batch_size=batch_size)
conf = np.zeros([len(classes),len(classes)])
confnorm = np.zeros([len(classes),len(classes)])
for i in range(0,X_test.shape[0]):
    j = list(Y_test[i,:]).index(1)
    k = int(np.argmax(test_Y_hat[i,:]))
    conf[j,k] = conf[j,k] + 1
for i in range(0,len(classes)):
    confnorm[i,:] = conf[i,:] / np.sum(conf[i,:])
cor = np.sum(np.diag(conf))
ncor = np.sum(conf) - cor
print("Overall Accuracy - Complex: ", cor / (cor+ncor))
acc = 1.0*cor/(cor+ncor)
plt.figure()
plot_confusion_matrix(confnorm, labels=classes)
```

```
Overall Accuracy - Complex:  0.4989363636363636
```

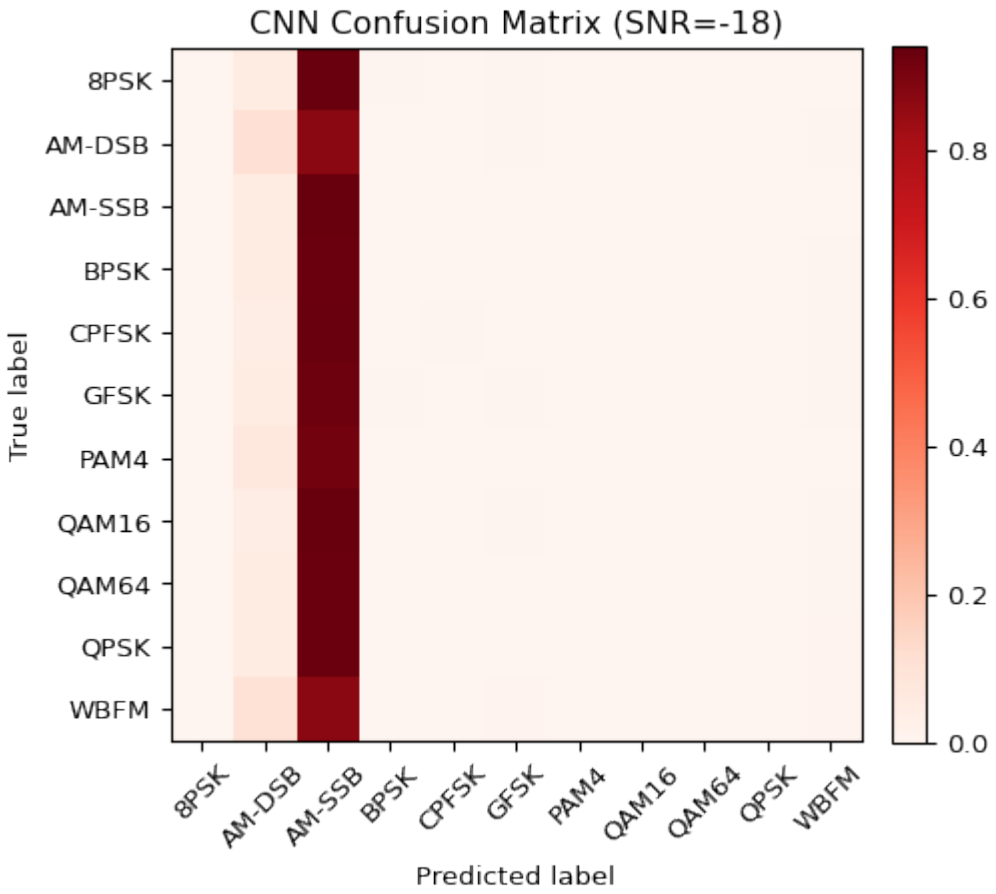
```
<Figure size 432x288 with 0 Axes>
```



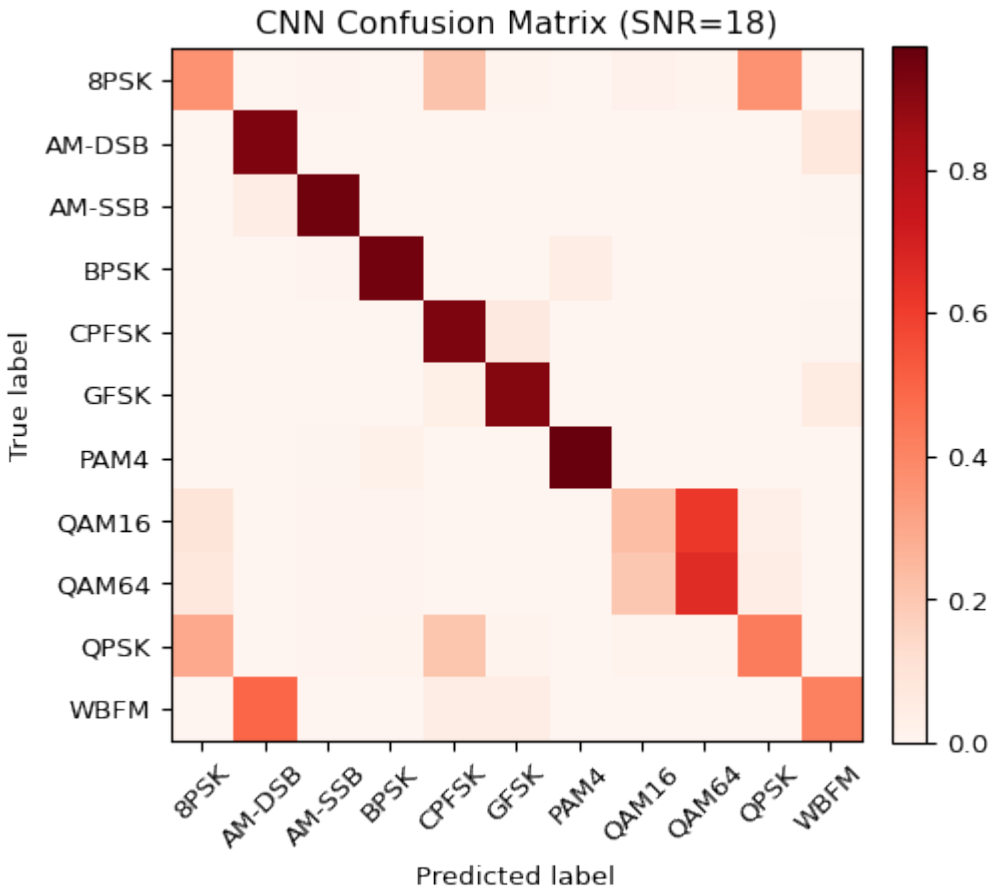
[15]:
```
#Creating one hot labels
labels_oh       = np.eye(11)
samples_db      = np.zeros((20, 11000, 2, 128))
truth_labels_db = np.zeros((20, 11000, 11))

#Pulling out the data by SNR
for i in range(len(test_snrs)):
    for j in range(len(mods)):
        samples_db[i, j*1000:(j+1)*1000,:,:]    = Xd[(mods[j],test_snrs[i])]
        truth_labels_db[i, j*1000:(j+1)*1000,:] = labels_oh[j]
```

[18]:
```
#Plotting confusion matrix for different SNR levels for the first CNN
acc_cnn = np.zeros(len(test_snrs))
for s in range(20):

    test_X_i = samples_db[s]
    test_Y_i = truth_labels_db[s]

    test_Y_i_hat = cnn.predict(test_X_i)
    conf = np.zeros([len(mods),len(mods)])
    confnorm = np.zeros([len(mods),len(mods)])
    for i in range(0,test_X_i.shape[0]):
        j = list(test_Y_i[i,:]).index(1)
        k = int(np.argmax(test_Y_i_hat[i,:]))
        conf[j,k] = conf[j,k] + 1
    for i in range(0,len(mods)):
        confnorm[i,:] = conf[i,:] / np.sum(conf[i,:])
    #print the confusion matrix @ -18dB and 18dB
    if s == 1 or s == 19:
        plt.figure()
        plot_confusion_matrix(confnorm, labels=classes, title="CNN Confusion␣
 ↪Matrix (SNR=%d)"%(test_snrs[s]))
    cor = np.sum(np.diag(conf))
    ncor = np.sum(conf) - cor
    acc_cnn[s] = 1.0*cor/(cor+ncor)
print(acc_cnn)
```

```
[0.09745455 0.09790909 0.10381818 0.11372727 0.13309091 0.17081818
 0.23481818 0.36918182 0.51690909 0.62618182 0.68754545 0.70181818
 0.71436364 0.70709091 0.71072727 0.70527273 0.70263636 0.71018182
 0.712      0.70227273]

<Figure size 432x288 with 0 Axes>
```



<Figure size 432x288 with 0 Axes>

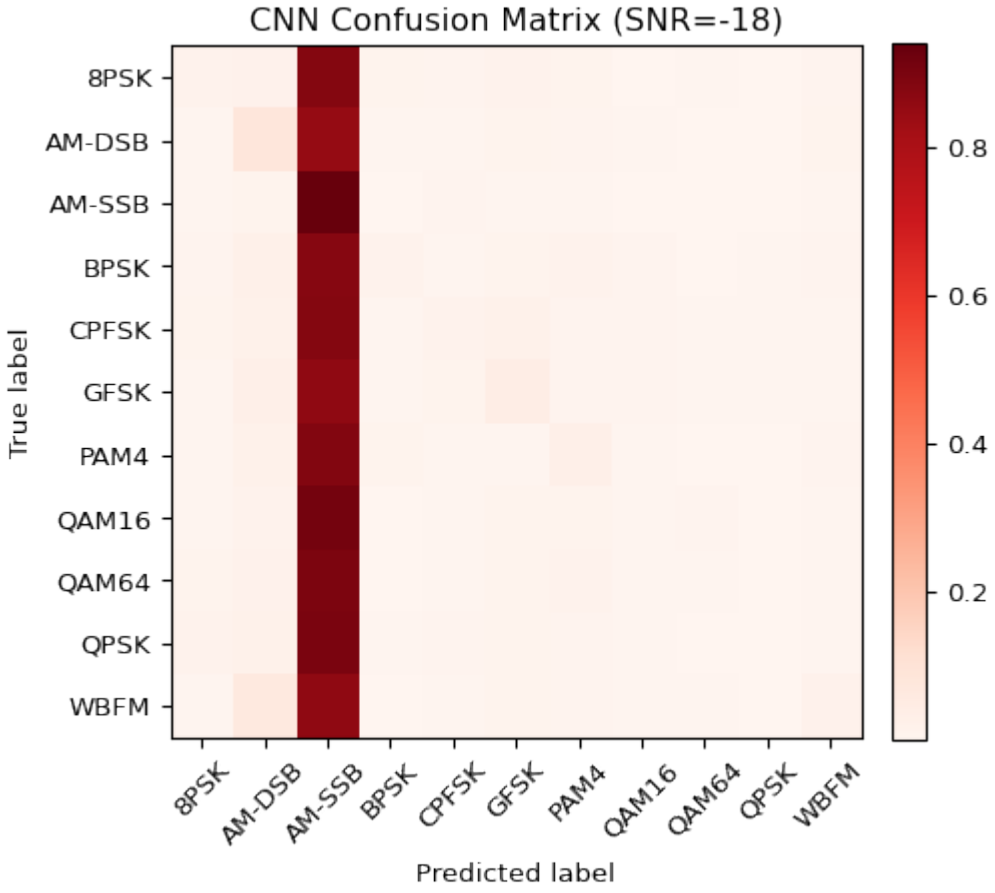


[19]:
```
#Plotting confusion matrix for different SNR levels for the second complex CNN
acc_complex = np.zeros(len(test_snrs))
for s in range(20):

    test_X_i = samples_db[s]
    test_Y_i = truth_labels_db[s]

    test_Y_i_hat = complex_CNN.predict(test_X_i)
    conf = np.zeros([len(mods),len(mods)])
    confnorm = np.zeros([len(mods),len(mods)])
    for i in range(0,test_X_i.shape[0]):
        j = list(test_Y_i[i,:]).index(1)
        k = int(np.argmax(test_Y_i_hat[i,:]))
        conf[j,k] = conf[j,k] + 1
    for i in range(0,len(mods)):
        confnorm[i,:] = conf[i,:] / np.sum(conf[i,:])
    #print the confusion matrix @ -18dB and 18dB
    if s == 1 or s == 19:
        plt.figure()
```

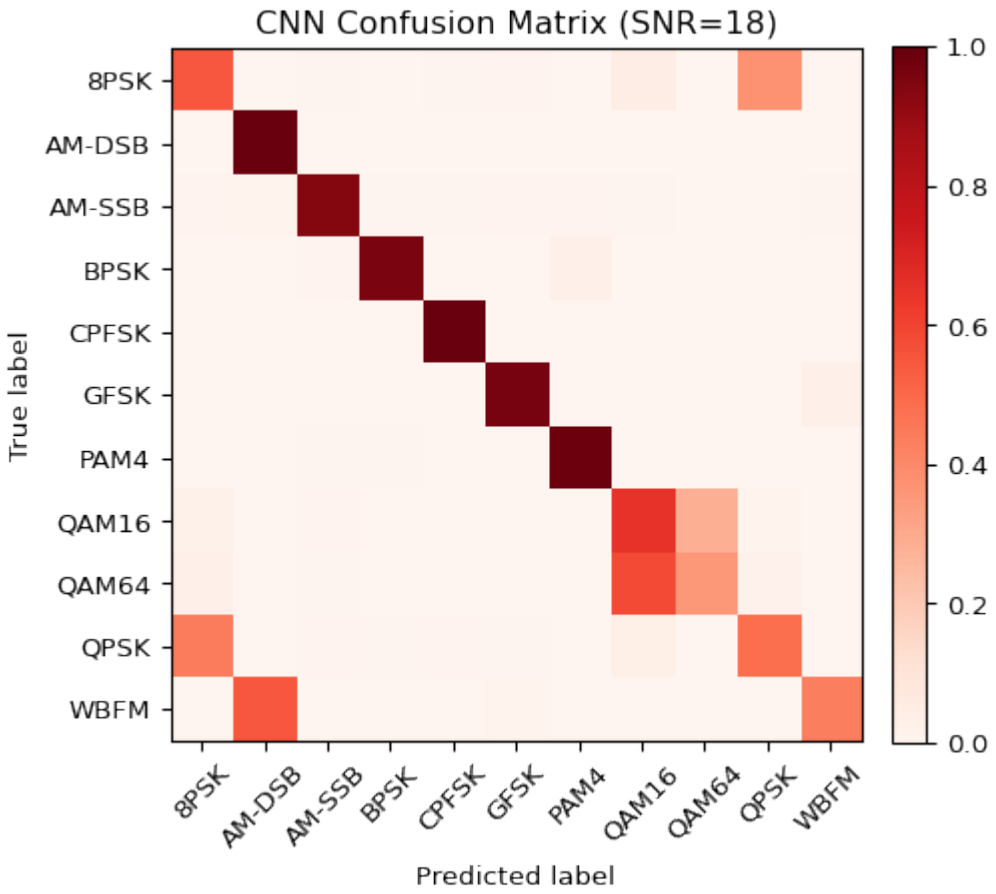
```
        plot_confusion_matrix(confnorm, labels=classes, title="CNN Confusion␣
 ↪Matrix (SNR=%d)"%(test_snrs[s]))
    cor = np.sum(np.diag(conf))
    ncor = np.sum(conf) - cor
    acc_complex[s] = 1.0*cor/(cor+ncor)
print(acc_complex)
```

```
[0.10727273 0.10754545 0.11627273 0.12827273 0.15381818 0.20790909
 0.31481818 0.46772727 0.59972727 0.69727273 0.74127273 0.75127273
 0.764      0.76190909 0.76263636 0.75945455 0.76190909 0.761
 0.76018182 0.75736364]
```

```
<Figure size 432x288 with 0 Axes>
```

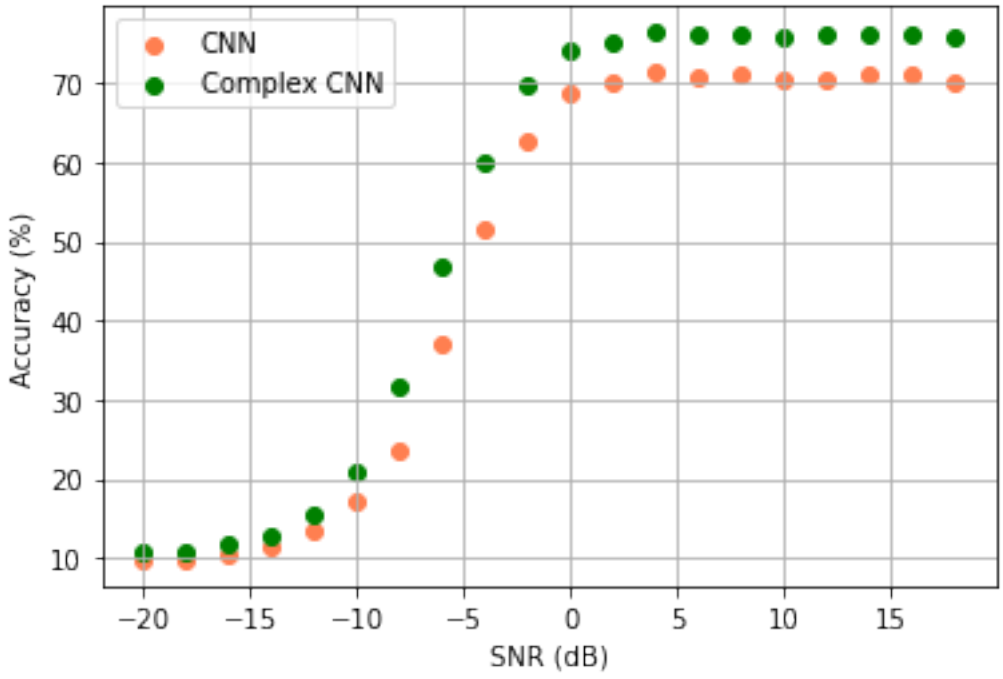



```
<Figure size 432x288 with 0 Axes>
```



[20]:
```
#plotting the classification accuracy with respect to SNR
plt.scatter(range(-20,20,2),acc_cnn*100, label = "CNN", color = 'coral')
plt.scatter(range(-20,20,2),acc_complex*100, label = 'Complex CNN', color =␣
↪'green')
plt.grid()
plt.xlabel('SNR (dB)')
plt.ylabel('Accuracy (%)')
plt.legend()
```

[20]:
```
<matplotlib.legend.Legend at 0x26a8cb624c0>
```


CNN
Complex CNN
Accuracy (%)
SNR (dB)
10
20
30
40
50
60
70
−20
−15
−10
−5
0
5
10
15